Lattice QCD package GWU-code and QUDA with HIP

Yu-Jiang Bi[*][†]
Institute of High energy Physics, Chinese Academy of Sciences, Beijing 100049, China

Yi Xiao
The Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
E-mail: louisxcode@gmail.com

Wei-Yi Guo
Department of Physics, University of Warwick, Coventry, CV4 7AL, United Kingdom

Ming Gong
Institute of High energy Physics, Chinese Academy of Sciences, Beijing 100049, China

Peng Sun
Nanjing Normal University, Nanjing, Jiangsu, 210023, China
E-mail: 06260@njnu.edu.cn

Shun Xu
Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China

Yi-Bo Yang
CAS Key Laboratory of Theoretical Physics, Institute of Theoretical Physics, Chinese Academy of Sciences, Beijing 100190, China
E-mail: ybyang@itp.ac.cn

The open source HIP platform for GPU computing provides an uniform framework to support both the NVIDIA and AMD GPUs, and also the possibility to porting the CUDA code to the HIP-compatible one. We present the porting progress on the Overlap fermion inverter (GWU-code) and also the general Lattice QCD inverter package - QUDA. The manual of using QUDA on HIP and also the tips of porting general CUDA code into the HIP framework are also provided.

37th International Symposium on Lattice Field Theory - Lattice2019
16-22 June 2019
Wuhan, China

[*]Speaker.
[†]This work is supported in part by the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDC01040100) and the National Key Research and Development Program of China (No. 2017YFB0203200). M. Gong is supported in part by the National Science Foundation of China (NSFC) under the project No. 11775229, P. Sun is supported by Natural Science Foundation of China under grant No. 11975127, Y. Yang is supported in part by Chinese Academy of Science CAS Pioneer Hundred Talents Program.

1. Introduction and background

The engineering and energy efficiency constraints push the modern supercomputer architecture to the multi-level parallelism, and heterogeneous computing architectures such as CPU+GPU are widely used in top500 supercomputers, including the most recent fastest two, Summit and Sierra. Most of the performance on such a computer come from the Nvidia GPU V100, and an efficient code is essential to benefit the related Lattice QCD calculation from those machines. There are already quite a few package can support the Nvidia GPU code platform CUDA with good performance and also multi-GPU scaling, including QUDA (for most of the fermion actions) [1, 2, 3], GRID (for the domain wall fermion and etc.) [4], GWU-code (for the overlap and clover fermion) [5, 6], and so on.

On the other hand, the efficient code on the AMD GPU falls behind except some effects with OpenCL (e.g., CL2QCD [7]), while the peak performance of the AMD GPU have caught up and the E-flops supercomputer “Frontier" with AMD GPU will be built in US by 2021. In recent years, an open source HIP platform is promoted to support both the Nvidia and AMD GPUs, which also provide the possibility to porting the CUDA codes to the HIP platform. Based on this, writing the code from scratch and implementing hundreds features needed by the lattice QCD calculation, would be avoid by porting the existed CUDA codes. In this proceeding, we will present our finding on using the package GWU-code and QUDA on the AMD GPU through the HIP platform, with a summary on the known issues.

2. Porting tips and compiling manual of QUDA

Generally, the porting can be separated into 3 stages, convert the code with hipify-perl, patch the codes manually to satisfy the requirement of compiler, replace the unsupported features to avoid the runtime crash. Let us taking the porting of QUDA as example:

1). Convert the code with hipify-perl. The hipify-perl is a perl script to map the name of the CUDA functions to that of their HIP counterpart, and also the CUDA head files. If the script meets some unknown words starting with "cu", message “warning:... : unsupported device function" will be thrown out, while the conversion will continue.

2). Patch the codes manually to satisfy the requirement of compiler. Currently, QUDA is compiled by hip-clang, instead of HCC (Heterogeneous Compute Compiler). HCC is HC compiler, which is C ++ AMP syntax language with HSA Extend [8]. HCC will translate HIP kernel syntax into C ++ AMP syntax by using functional or macro grid launch, while certain QUDA device function used complicated class and template will cause syntax or runtime error. In the other hand, hip-clang is a hip kernel syntax supported LLVM frontend as shown in Fig. 1. By setting the environment variable HIP_PLATFORM to clang, hip-clang will take over the compiling and compile CUDA-like syntax source code to LLVM IR directly [9], and then the AMD GPU backend of LLVM will compile the LLVM IR to binary. The patches we applied include:

2.1) Set CMAKE_CXX_SYSROOT_FLAG_CODE to add the .cu suffix to the CMAKE_CXX_SOURCE_FILE_EXTENSIONS, and then use hip-clang to compile both the .cpp and .cu files. Note that the flag “-g" should be avoid and “-_STRICT_ASNI_ -O3" is necessary

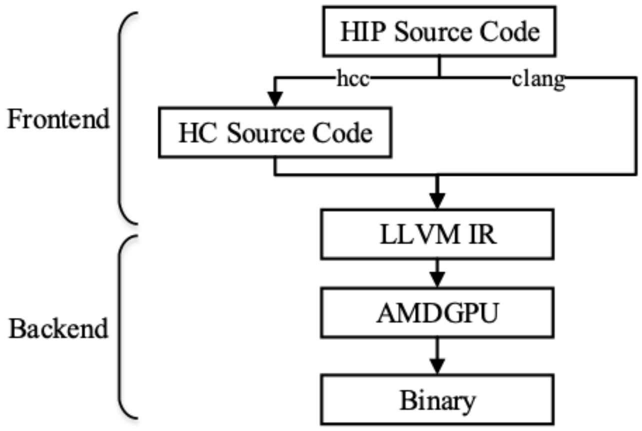

Figure 1: HIPCC Compilation Process. The clang compiler skips the step to generate the HC source code.

to make the code works well. Compile clover_deriv_quda.cu and gauge_stout.cu will crash the compiler and add the flag "-fno-inline" would be a choice to avoid it.

2.2) Fix the functions with different prototype, likes hipGetErrorString, hipGetErrorName, hipPointerGetAttributes, hipMemcpyHtoDAsync and etc. At the same time, some of the functions likes blasCgetrfBatched and blasCgetriBatched are replaced with the CPU version.

2.3) Add the __host__ flags in front of the __device __ functions which used in the CPU code.

2.4) Rewrite the ptx code into the normal C++ codes.

2.5) Move the declare of the shared memory into the body of the functions, as it can not be located in the link stage.

3). Replace the unsupported features to avoid the runtime crash. The major changes include:

3.2) Suppose the memoryType is host if hipPointerGetAttributes return an error, and comment out checkCudaError() in the constructors;

3.3) Limited the maximum threads used in the tuning to 512 or even smaller number, if certain function crashes on the AMD GPU with more threads.

3.4) Use the global function to copy the constant array needed by multi_blas_kernel to the global memory, as the function hipmemcpyToSymbol doesn't work correctly:

```
__global__ void set_Amatix(signed char *ref) {
        int idx = blockIdx.x * blockDim.x + threadIdx.x;
        if(idx>=MAX_MATRIX_SIZE)
                return;
        Amatrix_d[idx]=ref[idx];
}
```

```
        signed char *A_d;
        hipMalloc(&A_d, MAX_MATRIX_SIZE);
        hipMemcpy(A_d,A,MAX_MATRIX_SIZE,hipMemcpyHostToDevice);
        set_Amatix<<<256,MAX_MATRIX_SIZE/256>>>(A_d);
        hipDeviceSynchronize();hipFree(A_d);
```

Module name	Ported	Tested
QUDA_DIRAC_WILSON	yes	yes
QUDA_DIRAC_CLOVER	yes	yes
QUDA_CONTRACT	yes	yes
QUDA_COVDEV	yes	yes
QUDA_DIRAC_STAGGERED	yes	no
QUDA_FORCE_GAUGE	yes	no
QUDA_DIRAC_DOMAIN_WALL	no	no
QUDA_DIRAC_TWISTED_MASS	no	no
QUDA_DIRAC_TWISTED_CLOVER	no	no
QUDA_DIRAC_CLOVER_HASENBUSCH	no	no
QUDA_DIRAC_NDEG_TWISTED_MASS	no	no
QUDA_LINK_ASQTAD	no	no
QUDA_LINK_HISQ	no	no
QUDA_FORCE_HISQ	no	no
QUDA_GAUGE_TOOLS	no	no
QUDA_GAUGE_ALG	no	no
QUDA_DYNAMIC_CLOVER	no	no

Table 1: Summary of the porting progress in term of QUDA build options. All the parts needed for a Multigrid inverter of the Clover fermion have been done.

3.5) In the function multiblas, multireduce and ComputeVUV, pass the parameters though the argument class of the global function likes

```
__global__ func(Arg arg){...}
```

can make the performance to be extremely low. Such a problem can be avoid by copying the argument class to the GPU memory first, and use the its reference as the argument:

```
__global__ func(Arg &arg){...}
```

```
      Arg *arg_d;
      hipMalloc(&arg_d, sizeof(Arg));
      hipMemcpy(arg_d,&arg,sizeof(Arg),hipMemcpyHostToDevice);
      func(*arg_d);
      hipDeviceSynchronize();
      hipFree(arg_d);
```

Other minor hacks can be found in the present branch on the github:

```
      https://github.com/lattice/quda/tree/rocm-devel.
```

The present code merged the recent released QUDA 1.0.0. In term of the QUDA build options, the porting progress are listed in Table 1.

	Peak FP32 (TFlops)	Bandwidth (TB)	GWU-code (TFlops)	QUDA (TFlops)
Nvidia V100	14.7	0.9	0.80	1.11
AMD MI60	14 (for PCIe)	up to 1.0	0.54	0.48

Table 2: The single precision peak performance and memory bandwidth of V100 and MI60, v.s. the D-slash performances using the GWU-code and QUDA with the same single precision. The SU(3) is suppressed to 12 real numbers in both the GWU-code and QUDA cases to save the memory bandwidth. Note that the tests on V100 uses CUDA, not HIP.

3. GWU-code case

Comparing to QUDA, porting GWU-code is much simpler. GWU-code use the macro to generate the D-slash GPU kernel without any device functions, and implement the vector operator on GPU with the CUDA Thrust [5, 6]. Thus one just need to replace the Thrust library with the rocthrust after the code has been converted with hipify-perl. The function thrust::reduce can be very slow with double precision in certain version, but it can be replaced by the other functions with normal performance.

4. Performance and issues

Our test is based on AMD MI60 GPU and Nvidia V100 GPU. The D-slash performance is summarized in Table 2. The QUDA dslash performance is somehow lower as this kernel is much more complicated and then can not be fully optimized with hip-clang at present.

As the practical application of QUDA, the multigrid inverter works correctly while the performance is not very permising. With the largest $96^3 \times 192$ lattice we tested, the total performance with 324 MI60 GPUs is around 10 TFlops, using a 3-level multigrid layouts (4,4,4,4) and (2,2,2,2).

In the GWU-code side, 200 pairs of the Overlap eigensystem of the HYP smeared $24^3 \times 64$ RBC configuration at a=0.11fm can be generated with 4 MI60 GPU in 4 hours, and the similar calculation with E5-2698v3 at 2.3 GHz requires 1024 cores for 2 hours. The test with larger size is in progress.

5. Summary

In summary, we port two Lattice QCD CUDA packages, GWU-code and QUDA to the AMD GPU platform using HIP. The performance is around half of that on the CUDA when the memory bandwidth of them are similar. The multigrid inverter of QUDA works correctly with the lattice as large as $96^3 \times 192$, and the overlap eigensystem can be generated correctly with GWU-code. We will try to optimize the performance and scaling in the further study. All the present test using HIP are done on AMD GPUs, that on Nvidia GPUs will be also investigated.

References

[1] M. A. Clark, R. Babich, K. Barros, R. C. Brower, and C. Rebbi. Solving Lattice QCD systems of equations using mixed precision solvers on GPUs. *Comput. Phys. Commun.*, 181:1517–1528, 2010.

[2] R. Babich, M. A. Clark, B. Joo, G. Shi, R. C. Brower, and S. Gottlieb. Scaling Lattice QCD beyond 100 GPUs. In *SC11 International Conference for High Performance Computing, Networking, Storage and Analysis Seattle, Washington, November 12-18, 2011*, 2011.

[3] M. A. Clark, Blint Jo, Alexei Strelchenko, Michael Cheng, Arjun Gambhir, and Richard Brower. Accelerating Lattice QCD Multigrid on GPUs Using Fine-Grained Parallelization. 2016.

[4] Peter Boyle, Azusa Yamaguchi, Guido Cossu, and Antonin Portelli. Grid: A next generation data parallel C++ QCD library. 2015.

[5] A. Alexandru, C. Pelissier, B. Gamari, and F. Lee. Multi-mass solvers for lattice QCD on GPUs. *J. Comput. Phys.*, 231:1866–1878, 2012.

[6] Andrei Alexandru, Michael Lujan, Craig Pelissier, Ben Gamari, and Frank X. Lee. Efficient implementation of the overlap operator on multi-GPUs. In *Proceedings, 2011 Symposium on Application Accelerators in High-Performance Computing (SAAHPC'11): Knoxville, Tennessee, July 19-20, 2011*, pages 123–130, 2011.

[7] Owe Philipsen, Christopher Pinke, Alessandro Sciarra, and Matthias Bach. CL^{2}QCD - Lattice QCD based on OpenCL. *PoS*, LATTICE2014:038, 2014.

[8] http://www.hsafoundation.com/standards/.

[9] J Wu, A Belevich, and E Bendersky. an open-source gpgpu compiler. *Proceedings of the 2016 International Symposium on Code Generation and Optimization. ACM, 2016: 105-116.*, 2016.